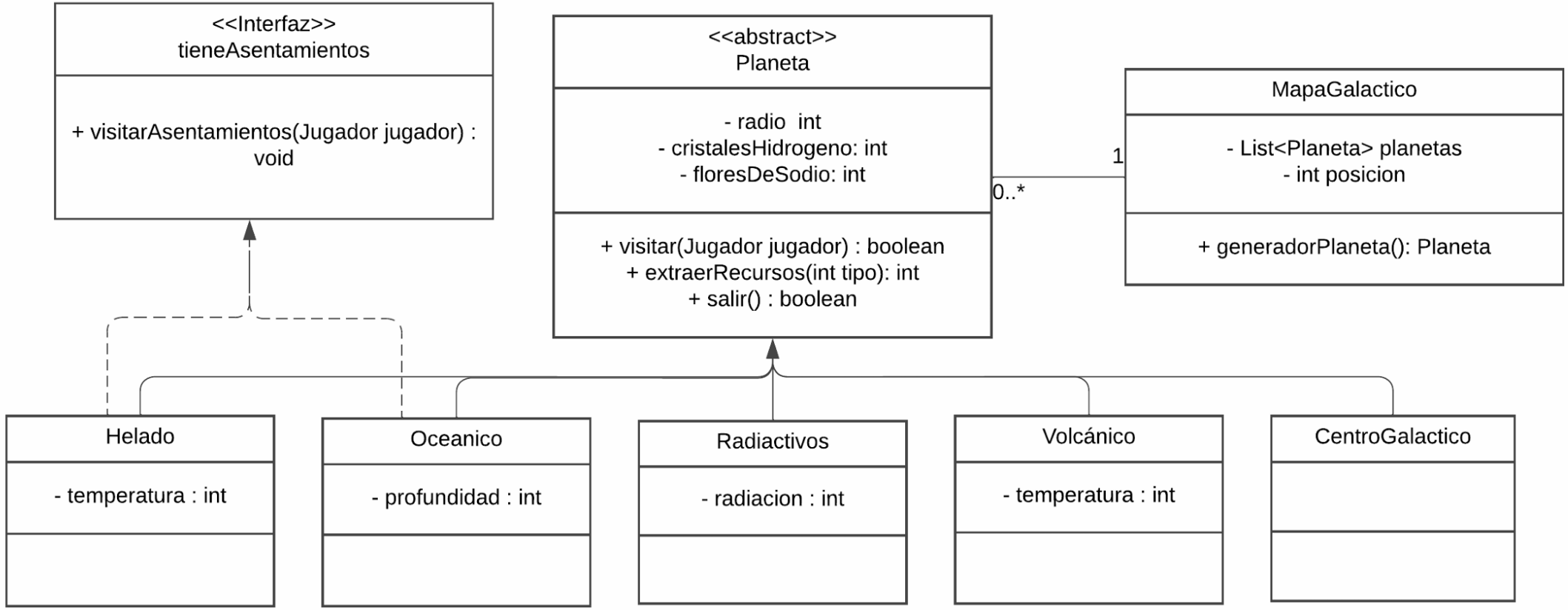
CÁPSULA:

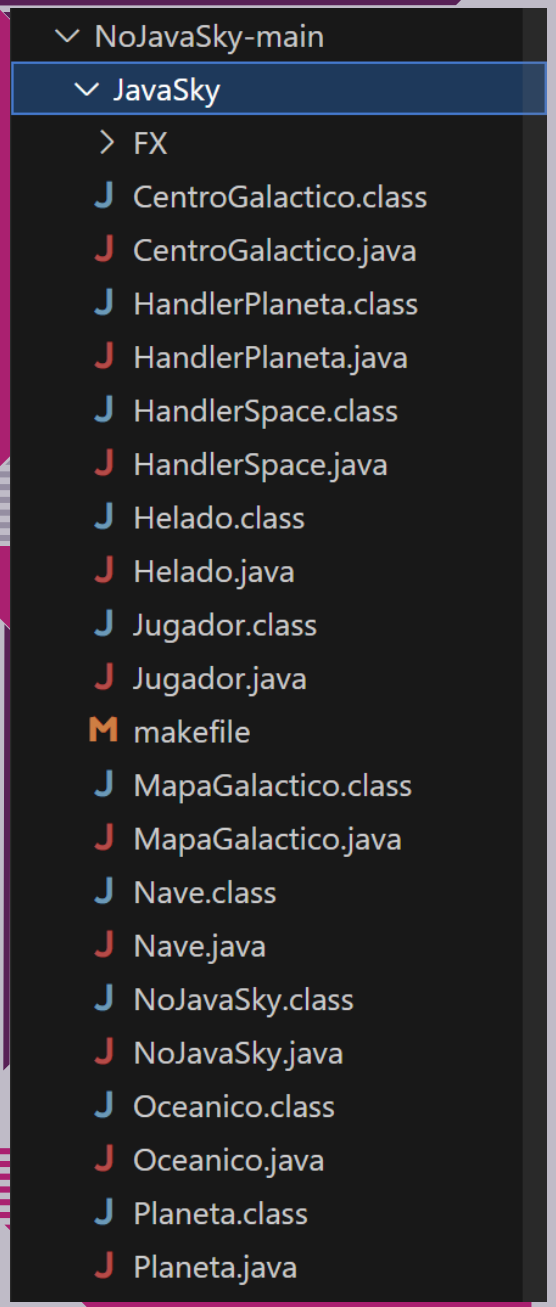
PROGRAMACIÓN ORIENTADA A OBJETOS

AYUDANTE: ELISMAR SUÁREZ

EJEMPLO DE UN DIAGRAMA DE CLASES UML



INTERFAZ



makefile ● | HandlerPlaneta.java ● | *tieneAsentamientos.java* ×

NoJavaSky-main > JavaSky > tieneAsentamientos.java

```
1
2   public interface tieneAsentamientos {
3       void visitarAsentamientos(Jugador jugador);
4       void comerciar(Jugador jugador) throws InterruptedException;
5   }
6
```

Es una colección de métodos abstractos (sin implementación) que **se definen en la interfaz** pero **se implementan (instancian) en las clases** que la usan. Por lo que obligatoriamente debe tener clases hijas.

No tiene constructores.

→ Los archivos .java pertenecen a la misma carpeta.

CLASE ABSTRACTA

makefile ● | HandlerPlaneta.java ● | Planeta.java ● | Helado.java ●

NoJavaSky-main > JavaSky > Planeta.java

```java

public abstract class Planeta {
    private int radio;
    private int cristalesDeHidrogeno;
    private int floresDeSodio;
    private String tipoPlaneta;


    public Planeta(int radio, int cristalesDeHidrogeno, int floresDeSodio, String tipoPlaneta) {
        this.radio = radio;
        this.cristalesDeHidrogeno = cristalesDeHidrogeno;
        this.floresDeSodio = floresDeSodio;
        this.tipoPlaneta = tipoPlaneta;
    }

```

Una **Clase Abstracta** es utilizada para crear plantillas de otras clases pues genera una estructura en común. Se caracteriza por ser **clase padre** en una relación de herencia.

- Pueden tener contructores
- Métodos abstractos
- Métodos concretos

GET & SET

NoJavaSky-main > JavaSky > Planeta.java

```java

public abstract class Planeta {
    private int radio;
    private int cristalesDeHidrogeno;
    private int floresDeSodio;
    private String tipoPlaneta;


    public Planeta(int radio, int cristalesDeHidrogeno, int floresDeSodio, String tipoPlaneta) {
        this.radio = radio;
        this.cristalesDeHidrogeno = cristalesDeHidrogeno;
        this.floresDeSodio = floresDeSodio;
        this.tipoPlaneta = tipoPlaneta;
    }


    public int getRadio() {
        return radio;
    }


    public void setRadio(int radio) {
        this.radio = radio;
    }
```

Fundamentales para la POO, principio de encapsulamiento.

- **Método get:** Se usa para **obtener** el valor de un atributo privado.
- **Método set:** Se usa para **modificar/establecer** el valor de un atributo privado.

avaSky-main > JavaSky > Planeta.java

```java
public abstract class Planeta {
    private int radio;
    private int cristalesDeHidrogeno;
    private int floresDeSodio;
    private String tipoPlaneta;


    public Planeta(int radio, int cristalesDeHidrogeno, int floresDeSodio, String tipoPlaneta) {
        this.radio = radio;
        this.cristalesDeHidrogeno = cristalesDeHidrogeno;
        this.floresDeSodio = floresDeSodio;
        this.tipoPlaneta = tipoPlaneta;
    }
```

Contructor de la clase planeta

this

- Es una **autorreferencia** al objeto actual dentro de una clase,
- Es útil para cuando un parámetro de método tiene el mismo nombre que un atributo de la clase, this elimina la ambigüedad

MODIFICADORES DE ACCESO EN JAVA

Modificador	**Descripción**
Private	sólo accesibles por la propia clase.
package	miembros sin modificador de acceso son sólo accesibles por código en el mismo paquete.
Protected	accesibles por una subclase, como también por código del mismo paquete.
Public	accesibles por cualquier clase.

makefile ● HandlerPlaneta.java ● Helado.java ●

NoJavaSky-main > JavaSky > Helado.java

```java
import java.util.Scanner;

public class Helado extends Planeta implements tieneAsentamientos {
    private int temperatura;
    private boolean enSuperficie = false;
    private double consumoEnergia;
    private Scanner scanner = new Scanner(System.in);
    private boolean primeraVisita;


    public Helado(int radio, int cristalesDeHidrogeno, int floresDeSodio, int temperatura) {
        super(radio, cristalesDeHidrogeno, floresDeSodio, "\u001B[36mHelado\u001B[0m");
        this.temperatura = temperatura;
        this.consumoEnergia = 0.15 * Math.abs(temperatura);
        this.primeraVisita = true;
    }


    public int getTemperatura() {
        return temperatura;
    }


    public void setConsumoEnergia(Jugador jugador){
        jugador.setConsumoEnergia(consumoEnergia);
    }
```

Las subclases pueden añadir atributos a los constructores heredados

Subclase de planeta, implementa la interfaz y extiende la superclase.

Por lo que hereda sus atributos y métodos.

super

Es una referencia a la **clase padre (superclase)** y se usa principalmente en contextos de herencia.

Debe ser la primera línea en un constructor de subclase. De no usarlo se hereda el constructor del padre.

Es utilizado para extender el comportamiento del padre y no tener que reescribirlo.

OVERRIDE VS OVERLOAD

Override (Sobrescritura):
Modificar un método de la clase padre en la clase hija, manteniendo su misma firma (nombre, parámetros y tipo de retorno).

@Override (buena práctica)

Overload (Sobrecarga):
Crear múltiples versiones de un método en la misma clase con diferentes parámetros. No mantiene la firma.

Puede cambiar el tipo de retorno.
No requiere herencia.